Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Theoretical Computer Science 299 (2003) 387–412

Theoretical Computer Science

www.elsevier.com/locate/tcs

Approximate pattern matching and transitive closure logics☆

Kjell Lemström[a,*,1], Lauri Hella[b,2]

[a] *Department of Computer Science, P.O. Box 26, University of Helsinki, FIN-00014, Helsinki, Finland*
[b] *Department of Mathematics, P.O. Box 4, University of Helsinki, FIN-00014, Helsinki, Finland*

Received 2 December 1999; received in revised form 1 October 2001; accepted 25 April 2002
Communicated by M. Nivat

Abstract

A sartorial query language facilitates the formulation of queries to a (string) database. One step towards an implementation of such a query language can be taken by defining a logical formalism expressing a known solution for the particular problem at hand. The simplicity of the logic is a desired property, because the simpler the logic that the query language is based on, the more efficiently it can be implemented. We introduce a logical formalism for expressing approximate pattern matching. The formalism uses properties of the *dynamic programming* approach; a *minimizing path* of a dynamic programming table is expressed by using a formula in an extension of first order logic (FO). We consider the well-known problems of *k-mismatches* and *k-differences*. Assuming first that k is given as a part of the input, those problems are expressed by using deterministic transitive closure logic (FO(DTC)) and transitive closure logic (FO(TC)), respectively. We show how to adapt the formalisms to allow individual costs for the *editing operations*, and consider music information retrieval (MIR) as a case study.

We believe that in the general case k-differences is not expressible in FO(DTC). However, we show that proving this is at least as hard as separating LOGSPACE from NLOGSPACE. On the other hand, we show that if k is fixed, the k-differences problem can be expressed by an FO(DTC) formula.

Keywords: Logic in computing; Combinatorial pattern matching; Music information retrieval

☆ A preliminary version of this paper has appeared in [12].

* Corresponding author.
E-mail address: klemstro@cs.helsinki.fi (K. Lemström).

[1] Partially supported by the Helsinki Graduate School of Computer Science and Engineering.
[2] Supported in part by grant #28139 from the Academy of Finland. Current address: Department of Mathematics, Statistics and Philosophy, Kanslerinrinne 1, 33014 University of Tampere, Finland.

0304-3975/03/$ - see front matter
PII: S0304-3975(02)00484-X

1. Introduction

General approximate string matching has been well studied during the last couple of decades (for an exposition of the basic methods and algorithms we refer to [2,8]). The algorithms have been applied to diverse research fields. Multimedia information retrieval is one active and wide area which is applying these methods for locating and retrieving information in large databases, such as biosequence (e.g. [1]), image (e.g. [17]) and music databases (e.g. [11]). *Edit distance* is a widely used metrics for exposing the dissimilarity between two strings. It measures how many *editing operations* are needed to convert string A to be similar to string B. The calculation of edit distance can be done, e.g., by using *dynamic programming*. Given a constant k ($k \geqslant 0$) that is a threshold value to control the accuracy of the matching, and two strings P and S, the dynamic programming method is straightforwardly adaptable to locate *k-similar* occurrences of *pattern* P within *text* (or *source*) S.

In order to express approximate string matching, we present a formalism based on *transitive closure logic* (FO(TC)). FO(TC) is obtained by adding an operation that forms the transitive closure of definable relations, to first order logic (FO). In a special case of the approximate matching we are able to use *deterministic transitive closure logic* (FO(DTC)), which is a strictly weaker logic. The data complexities for FO(TC) and FO(DTC) are NLOGSPACE and LOGSPACE, respectively. Thus, for any fixed FO(TC) or FO(DTC) sentence there is a *Turing machine*, which takes a finite model as an input. Moreover, the machine can decide whether or not the FO(TC) or FO(DTC) sentence is true in the input model by using, respectively, nondeterministic or deterministic, logarithmic space. On ordered models the converse is also true; thus, e.g., every NLOGSPACE computable property is expressible in FO(TC).

In this study, we concentrate on the problem of finding approximate patterns in texts. In our formalism, a formula of transitive closure logic is used for expressing one of the minimizing paths that leads to an *approximate occurrence*, that is, a substring of the text such that at most k editing operations are needed to obtain it from the pattern. If the only allowed editing operation is *replacing* (the k-mismatches case), the formula is in FO(DTC) and is straightforward to find. However, the problem becomes somewhat more challenging if *deleting* and *inserting* operations are also allowed (the k-differences case): If k is given as a part of the input, the matching is expressed by using an FO(TC) formula. It is not certain, however, that this problem could not be expressed by FO(DTC), but proving that such a formula does not exist is at least as difficult as to show that NLOGSPACE $\neq$ LOGSPACE. On the other hand, if k is fixed (i.e., not part of the input), we can prove that the matching is expressible in FO(DTC).

We derive upper and lower bounds for the complexity of the approximate pattern matching problems. In particular, we show that the k-differences case of the problem can be solved by a Turing machine in $O((\log n)^2)$ space. The k-mismatches problem is solvable in $O(\log n)$ space. These upper bounds hold in the usual complexity theoretic setting in which instances are strings that are directly used as inputs of Turing machines. Changing to our model theoretic formalism in which instances are models, we obtain a tightly matching lower bound for the k-mismatches case: there is a LOGSPACE

complete problem that is reducible to the k-mismatches problem. However, this lower bound result is sensitive to the way the mismatches problem is formulated in the model theoretic setting: we show that after a small modification in the formalism, the k-mismatches problem can be expressed in first order logic with counting quantifiers, FO(COUNT), and hence the problem is in the complexity class ThC^0 (the class of problems computable by uniform sequences of polynomial size-constant depth circuits with threshold gates).

Assessing the complexity of approximate pattern matching problems and finding the simplest logics to express them, are already interesting in themselves. However, expressing the problems in a logic can be seen as a step towards constructing a string database query language (see e.g. [7,16]). To obtain the expressibility in transitive closure logics is particularly useful, because the relation algebra **A** by Date and Darwen [3], and the SQL:1999 standard [5] both comprise an explicit transitive closure operator. Therefore, the formalism can be used in developing a query language for a database management system. This approach may be useful with very large databases, where *indexing approaches* require an excessive amount of space and *online approaches* become too slow to use in practice. For instance, consider our case study, i.e. music information retrieval (MIR). The present amount of MIDI files [14] available on the Internet is too much to be stored in a *suffix tree* [23], and for a very efficient online retrieval algorithm using the fastest PCs presently available, it takes almost 1 min to scan through the strings generated out of those files [11]. Although that performance is actually not too bad, the situation shall be different due to an explosive increase of musical files on the Internet.

The rest of this paper is organized as follows. We continue in the next section with a brief presentation of the concepts and tools that are essential in the formalism: edit distance, dynamic programming, first order logic, transitive closure logic, and first order logic with counting. In Section 3, we first present a nondeterministic version of the k-differences algorithm, which shall be the starting point to our formalisms, and then define our model. In Sections 4 and 5 we consider cases where k is given as a part of the input, and study how the logic tools should be used to express the k-mismatches and k-differences problems, respectively. Section 6 describes how the k-differences problem can be expressed by FO(DTC) logic if the k is fixed in advance. Section 7 gives estimations for the complexity of finding patterns using string matching techniques. In Section 8, we illustrate the flexibility of the formalism. By considering MIR as a case study, we adapt the formalism to meet its special requirements. Finally, we conclude the paper in Section 9.

2. Concepts and tools

2.1. Edit distance

Let Σ be a finite set of symbols, the *alphabet*. Then any $A = a_1 \cdots a_m$ where each a_i is a symbol in Σ, is a *string* over Σ. The *length* of A is $|A| = m$. The string of length 0 is called the *empty sequence* and denoted λ. The set of all strings over Σ is denoted

by Σ^*. If a string A is of form $A=\beta\alpha\gamma$, where $\alpha,\beta,\gamma\in\Sigma^*$, we say that α is a *factor*, β a *prefix*, and γ a *suffix* of A. By $pre_j(A)$ we denote the prefix β of A such that $|\beta|=j$. A string A' is a *subsequence* of A if it can be obtained from A by deleting zero or more symbols, i.e., $A'=a_{i_1}a_{i_2}\cdots a_{i_m}$, where $i_1\ldots i_m$ is an increasing sequence of indices. The frequently used *editing operations* are as follows:

E1 *Replacing* the symbol a_j by another symbol b gives: $a_1\cdots a_{j-1}b\,a_{j+1}\cdots a_m$,
E2 *Inserting* a symbol b at position j gives: $a_1\cdots a_j b\,a_{j+1}\cdots a_m$,
E3 *Deleting* the symbol a_j at position j gives: $a_1\cdots a_{j-1}a_{j+1}\cdots a_m$,
E4 *Transposing* two symbols a_ja_{j+1} gives: $a_1\cdots a_{j-1}a_{j+1}a_ja_{j+2}\cdots a_m$.

In the edit distance framework, the general distance, denoted here as $D(A,B)$, is defined as follows.

Definition 1. The edit distance $D(A,B)$ is the minimum number of allowed editing operations required to convert string A into string B.

Numerous adaptations of the edit distance framework have been suggested. For instance, the well-known *unit cost edit distance* $D_L(A,B)$ (known also as Levenshtein distance) allows operations E1–E3. Other adaptations include, e.g., the *generalized Levenshtein distance* ($D_G(A,B)$) that allows operations E1–E4, and *Hamming distance* ($D_H(A,B)$) allowing only editing operation E1. Henceforth $D(A,B)$ shall denote collectively all these three distances.

Another way to view edit distances is to split A and B into equally many parts, where some of the parts may be empty sequences. Let $A=\alpha_1\ldots\alpha_p$ and $B=\beta_1\ldots\beta_p$ be such a splitting (also called a *trace*). Every trace suggests a way to obtain the string B from A by using *local transformations* of form $(\alpha_i\mapsto\beta_j)$. The local transformations comprise the allowed editing operations and an identity (ID) operation (which can be used for free):

ID Identity: a_j is preserved: $a_j=b_i$.

Thus, operations E1–E4 and ID are denoted as $a_j\mapsto b_i(a_j\neq b_i)$, $\lambda\mapsto b_i, a_j\mapsto\lambda$, $a_ja_{j+1}\mapsto a_{j+1}a_j$ and $a_j\mapsto b_i(a_j=b_i)$, respectively. A trace using the minimum number of editing operations (of all possible traces) gives the value for the edit distance $D(A,B)$. Moreover, the value equals the number of those required editing operations.

2.2. Dynamic programming and k similarity

In this paper, we are interested in the problem of finding k *similar* occurrences of a given pattern P ($P=p_1\ldots p_m$) within text S ($S=s_1\ldots s_n$), where k ($k\geqslant 0$) denotes the maximum allowed distance between the pattern and its occurrence in S. Typically, the pattern is very short as compared with the length of the text, that is, $m\ll n$.

Definition 2. Pattern P has a k similar occurrence in S if there is a factor P' of S such that $D(P,P')\leqslant k$.

The pattern matching problems associated with the Hamming and Levenshtein distances are called k-mismatches and k-differences, respectively. Therefore, when searching occurrences with k-mismatches, only E1 and ID should be allowed to be used, while in the case of k-differences the use of the operations E2 and E3 should be permitted, as well.

The occurrences of P within text S can be located by using dynamic programming technique [20]. Dynamic programming tabulates edit distances between all the possible prefixes of A and any factor of B: $d_{ij} = \min\{D(pre_j(A), b_r \dots b_i) \mid 1 \leqslant r \leqslant i+1\}$. Examples of such calculations can be found in Figs. 1 and 2, corresponding to the cases of k-mismatches and k-differences, respectively. Letting $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$, the k-differences algorithm is as follows:

Algorithm 1 (k-**differences**)
begin
1. **for** $j := 0$ **to** m **do** $d_{0j} = j$;
2. **for** $i := 1$ **to** n **do begin**
3. $\quad d_{i0} := 0$;
4. $\quad$ **for** $j := 1$ **to** m **do**
5. $\quad\quad d_{ij} := \min\{d_{i-1,j-1} + (\textbf{if } p_j = s_i \textbf{ then } 0 \textbf{ else } 1),$
$\quad\quad\quad\quad d_{i-1,j} + 1, d_{i,j-1} + 1\}$;
6. $\quad$ **if** $d_{im} \leqslant k$ **then** `write`(d_{im}, i);
7. **end**
end

Note that it is very easy to modify the algorithm above so that any (integer) costs could be assigned to the local transformations.

A *backtracking path* in a dynamic programming table starts from row m and ends at row 0 (see Figs. 1 and 2), and advances step by step (possibly not unique) according to the minimizing operation (line 5 of Algorithm 1). A *path* is a backtracking path whose direction is reversed.

Definition 3 (Minimizing path). A minimizing path is a path that ends in cell $d_{n'm}$ and $d_{n'm} = \min\{d_{im} \mid 1 \leqslant i \leqslant n\}$.

Theorem 4 (Sellers [19]). *Let there be a* (*minimizing*) *path ending in cell* d_{im} *and starting from* d_{r0}. *The edit distance* $D(P, s_{r+1} \dots s_i) = d_{im}$ *and* $d_{im} = \min\{D(P, s_t \dots s_i) \mid t \leqslant i+1\}$.

The approximate occurrences of P within S can be found by observing the bottom row of the dynamic programming table. Due to Theorem 4 above, the value of a cell d_{im} in the bottom row represents the minimum number of allowed editing operations used to convert P to be similar to any suffix of $pre_i(S)$. We call a path ending in cell d_{im} *accepting* if $d_{im} \leqslant k$.

If the starting index of an accepting path is needed, a procedure that resolves the path has to be called in the **if** statement on line 6 of Algorithm 1. Clearly, both the time and

space requirements (with respect to the RAM model [22]) for this basic algorithm are $O(mn)$. In an application where the exact starting point of an occurrence is not needed, it is enough to maintain space for $m+1$ cells of the dynamic programming table; the values in the previous column are always overwritten by the values of the column at hand as the calculation proceeds column-by-column. The cell above the adjacent cell in the previous column (corresponding to the replacing or identity) should be handled with care, and stored to a temporary variable. Moreover, the time requirement can also be diminished; it has been subsequently refined, first to $O(kn)$ expected time [21], and then to $O(kn)$ worst-case time [10]. [3]

2.3. First order logic

Vocabularies $\pi, \sigma, \ldots$ are finite sets consisting of *relation symbols* $R_0, R_1, \ldots$, and *constant symbols* $c_0, c_1, \ldots$. Every relation symbol R_i has a special natural number n_i that is attached to it, namely its *arity*. A *structure* (*model*) $\mathscr{A}$ of a vocabulary π is a tuple:

$$\mathscr{A} = \langle A, R_0^{\mathscr{A}}, R_1^{\mathscr{A}}, \ldots, c_0^{\mathscr{A}}, c_1^{\mathscr{A}}, \ldots \rangle,$$

where the nonempty set A is the *universe* of $\mathscr{A}$. Furthermore, for every relation symbol R_i in π, $R_i^{\mathscr{A}}$ is an n_i-ary relation on A, and for every constant c in π, $c_i^{\mathscr{A}}$ is an element of A.

A structure $\mathscr{A}$ is *finite* if its universe is a finite set. All structures considered in this paper are assumed to be finite.

The set of first order formulas over a vocabulary π is defined, as usually, by starting with atomic formulas and closing under negations $\neg\varphi$, disjunctions $\varphi \vee \psi$, and existential quantifications $\exists x\varphi$. An atomic formula over π is an expression of the form $R(t_1, \ldots, t_n)$ or $(t_1 = t_2)$, where $R \in \pi$ is an n-ary relation symbol and $t_1, \ldots, t_n$ are *terms*, i.e. either variables or constant symbols, in π. Other connectives (conjunction $\wedge$, implication $\rightarrow$, equivalence $\leftrightarrow$) and the universal quantifier ($\forall$) are defined in terms of $\neg$, $\vee$ and $\exists$ in the obvious way.

An occurrence of a variable x is *bound* in a formula φ if it is in the scope of some quantifier $\exists x$. If the variable x has an occurrence that is not bound, then x is a *free variable* of φ. The free variables of formulas are often explicitly displayed: we write $\varphi = \varphi(\bar{x})$ if the free variables of φ are in the tuple $\bar{x}$. A formula without free variables is a *sentence*.

The semantics of first order logic is defined in the standard way using Tarski's truth definition. If $\varphi(\bar{x})$ is a formula over a vocabulary π, $\mathscr{A}$ is a π-structure, and $\bar{a}$ is a tuple of elements of A such that $\varphi(\bar{x})$ is true in $\mathscr{A}$ when the variables in $\bar{x}$ are interpreted by the corresponding components of $\bar{a}$, then we write $\mathscr{A} \models \varphi[\bar{a}]$.

[3] The time complexity can be further refined by applying the hardware word-level parallelism of bit-vector operations. One of the most recent and fastest algorithms of such an approach runs in time $O(n\lceil m/w \rceil)$ [15], where w denotes the size of the computer word in bits.

2.4. Transitive closure logic

Next we give a brief review of logics obtained by augmenting first order logic with transitive closure operators. For more detailed expositions, we refer the reader to the books [4,9]. We start by defining the transitive closure and deterministic transitive closure operators TC and DTC. Let $h \geqslant 1$ and R be a $2h$-ary relation on a set A. Now $\mathrm{TC}(R)$ is defined as follows:

$$\mathrm{TC}(R) := \{(\bar{a},\bar{b}) \in A^{2h} \mid \text{there exist } n > 0 \text{ and } \bar{d}_0,\ldots,\bar{d}_n \in A^h \text{ such that } \bar{a} = \bar{d}_0, \bar{b} = \bar{d}_n, \text{ and for all } i < n,\ (\bar{d}_i,\bar{d}_{i+1}) \in R\}.$$

Thus, $\mathrm{TC}(R)$ consists of all pairs $(\bar{a},\bar{b})$ such that there is an R-path from $\bar{a}$ to $\bar{b}$ (viewing R as a directed graph on h-tuples of A). The deterministic transitive closure $\mathrm{DTC}(R)$ of R is defined similarly, except that branching of the R-path is disallowed:

$$\mathrm{DTC}(R) := \{(\bar{a},\bar{b}) \in A^{2h} \mid \text{there exist } n > 0 \text{ and } \bar{d}_0,\ldots,\bar{d}_n \in A^h \text{ such that } \bar{a} = \bar{d}_0, \bar{b} = \bar{d}_n, \text{ and for all } i < n, \bar{d}_{i+1} \text{ is the unique } \bar{d} \text{ for which } (\bar{d}_i,\bar{d}) \in R\}.$$

Transitive closure logic, FO(TC), is the extension of first order logic that is obtained by adding the new formula formation rule:

- if φ is a formula then $[TC_{\bar{x},\bar{y}}\varphi](\bar{u},\bar{v})$ is a formula,

where $\bar{x},\bar{y},\bar{u}$ and $\bar{v}$ are tuples of variables of the same length, and the variables in $\bar{x},\bar{y}$ are pairwise distinct. Here the operator $TC_{\bar{x},\bar{y}}$ binds all occurrences of the variables $\bar{x}$ and $\bar{y}$ in φ, whereas the variables in $\bar{u}$ and $\bar{v}$ are free in the new formula.

The semantics of FO(TC) is defined by adding the following rule to the truth definition of first order logic:

- $\mathscr{A} \models [TC_{\bar{x},\bar{y}}\varphi][\bar{a},\bar{b}]$ if and only if $(\bar{a},\bar{b}) \in \mathrm{TC}(\varphi^{\mathscr{A}})$, where $\varphi^{\mathscr{A}} = \{(\bar{c},\bar{d}) \mid \mathscr{A} \models \varphi[\bar{c},\bar{d}]\}$.

(Here we have suppressed free variables of φ other than those in $\bar{x}$ and $\bar{y}$.)

Deterministic transitive closure logic, FO(DTC), is defined in the same way as FO(TC), except that in the syntax the operator $TC_{\bar{x},\bar{y}}$ is replaced by $DTC_{\bar{x},\bar{y}}$, and the semantics is given by the deterministic transitive closure of relations.

It is well known that FO(DTC) and FO(TC) are contained in the complexity classes LOGSPACE and NLOGSPACE, respectively. In other words, for every FO(DTC) sentence there is a LOGSPACE Turing machine which takes a model as an input, and decides the truth of the sentence in the given model. Similarly, there is an NLOGSPACE Turing machine for every FO(TC) sentence. Furthermore, on ordered finite models FO(DTC) captures LOGSPACE and FO(TC) captures NLOGSPACE (see, e.g., [4,9]): if the vocabulary of the models considered contains a special binary relation symbol which is always interpreted as a linear order, then every LOGSPACE computable (NLOGSPACE computable) property is definable in FO(DTC) (FO(TC), resp.). Actually, it suffices that the models are equipped with a successor relation, since the corresponding linear order is easily definable by a formula of FO(DTC) from its successor relation.

2.5. First order logic with counting

Another way of increasing the expressive power of first order logic is to add counting mechanisms to it. First order logic with counting, FO(COUNT), is usually defined on two-sorted structures where the second sort consists of an initial segment $\{0,\ldots,n-1\}$ of the natural numbers equipped with order, addition and multiplication. We can also assume that each number $0,1,\ldots,n-1$ is given as a constant on the second sort since they are definable from the order. The length n of the second sort is equal to the number of elements in the first sort (see [9]). Counting power is introduced by the new formula formation rule:

- if $\varphi(x)$ is a formula then $\exists^i x\varphi(x)$ is a formula,

where x is a variable over the first sort, and i is a variable over the second (numerical) sort. Note that the counting quantifier $\exists^i x$ binds x, whereas i becomes a new free variable in the formula $\exists^i x\varphi(x)$. The semantics of the counting quantifiers is given by the rule:

- $\mathscr{A} \models \exists^i x\varphi(x)$ if and only if the number of elements a in the first sort of $\mathscr{A}$ such that $\mathscr{A} \models \varphi[a]$ is at least i.

Using the arithmetic functions on the numerical sort, FO(COUNT) can express all kinds of counting properties. For example, the sentence

$$\exists i\ (\exists k\ (i = k + k) \wedge \exists^i x R(x) \wedge \forall j\ (j = i + 1 \rightarrow \neg\exists^j x R(x)))$$

says that the relation R contains an even number of elements. We can also write a formula $\exists^{=i} x\varphi(x)$ saying that the number of elements x satisfying $\varphi(x)$ is exactly i:

$$\exists^{=i} x\varphi(x) := \exists^i x\varphi(x) \wedge \forall j(j = i + 1 \rightarrow \neg\exists^j x\varphi(x))).$$

The logic FO(COUNT) is also known to capture a complexity class; every problem expressible in FO(COUNT) is in ThC^0, and on ordered models $\mathrm{FO(COUNT)} = \mathrm{ThC}^0$ (see [9]). Here ThC^0 is the class of all problems computable by uniform sequences of polynomial size constant depth circuits with threshold gates. The class ThC^0 is contained in LOGSPACE, but it is not known whether the containment is strict.

3. Setting up the framework

Let $P = p_1 \ldots p_m$ and $S = s_1 \ldots s_n$ be the pattern and text, respectively. Without loss of generality we assume in the sequel that $n \geqslant m$. We often omit parentheses in formulas when there is no danger of incorrect interpretation.

3.1. Nondeterministic k-differences algorithm

Let us assume a nondeterministic function `guess`(p_1, p_2) that gets as input the lower and upper bounds of a subset of $\mathbb{N}$. Function `guess`(p_1, p_2) picks up a value between

the two integers p_1 and p_2 (inclusively) and returns it. The nondeterministic version of Algorithm 1 is as follows:

Algorithm 2 (**nondeterministic** k**-differences**)
begin
1. $i := \texttt{guess}(0, n)$;
2. $j := 0$; $count := 0$;
3. **repeat**
4. $direction := \texttt{guess}(1, 3)$;
5. **if not** $(direction = 2)$ **then** $j := j + 1$;
6. **if not** $(direction = 3)$ **then** $i := i + 1$;
7. **if not** $((direction = 1)$ **and** $(p_j = s_i))$ **then**
8. $count := count + 1$;
9. **until** $(j = m)$ **or** $(\text{count} > k)$;
10. **if** $count \leqslant k$ **then** *write* $(count, i)$;
end

Firstly on line 1, `guess` is a function that returns the location where an accepting path starts. The cost of required editing operations are accumulated to the variable *count* and the value of *direction* corresponds to one of the possible step directions of a local transformation in the dynamic programming table: identity and replacing correspond to a diagonal step ($direction = 1$), inserting to a horizontal ($direction = 2$), and deleting to a vertical step ($direction = 3$). One should be careful with the **if not** statements. They are used in order to avoid **if–else** structures. Thus they make the analysis shorter.

Let us assume that $k < m$, then the algorithm cannot execute the **repeat–until** loop more than $m + k + 1$ times (corresponding to m identities and $k + 1$ deleting and inserting operations). Therefore, for the nondeterministic Algorithm 2 the worst-case time requirement with respect to the RAM model is $O(m)$, while the space requirement is reduced to $O(1)$.

3.2. Table models

In order to obtain a model theoretic framework for expressing approximate pattern matching problems, we first have to encode their inputs (i.e., the strings P and S, and the error limit k) as models over some suitable vocabulary π. There is a standard way of encoding strings in Σ^* into models over a vocabulary consisting of a binary relation symbol Sc, and a unary relation symbol P_a for each $a \in \Sigma$: a string $B = b_1 \dots b_n$ is encoded by the model $\mathscr{A}_B = \langle [n], Sc, (P_a^B)_{a \in \Sigma} \rangle$, where $[n] = \{1, \dots, n\}$, $Sc = \{(i, i + 1) \mid 1 \leqslant i < n\}$ is the successor relation on $[n]$, and $P_a^B = \{i \mid b_i = a\}$.

Note that in this encoding the vocabulary of the models $\mathscr{A}_B$ depends on the alphabet Σ. However, for our purposes it is better to use a more uniform encoding. Hence we shall represent the symbols in Σ by elements of the model, which means that the strings P and S can be naturally represented as (partial) functions of the model.

More precisely, given an alphabet Σ, we fix a bijective mapping σ from Σ to an initial segment of $\mathbb{N}$ (thus, σ is an enumeration of the symbols in Σ). The **table model** $\mathscr{A}_{P,S,k}$ for pattern P, text S and error limit k is the model $\langle A, Sc, \mathrm{f_p}, \mathrm{f_s}, \mathrm{k}\rangle$ over a vocabulary with three binary relation symbols and one constant symbol, where the following hold:

$A=\{0,1,\ldots,r\}$ where r is the maximum of n and $\sigma(a)$, $a\in\Sigma$,
Sc is the successor relation on A,
$\mathrm{f_p}=\{(1,a_1),\ldots,(m,a_m)\}$ where $a_1=\sigma(p_1),\ldots,a_m=\sigma(p_m)$,
$\mathrm{f_s}=\{(1,b_1),\ldots,(n,b_n)\}$ where $b_1=\sigma(s_1),\ldots,b_n=\sigma(s_n)$,
k is the allowed maximum for edit distance.

(We assume here without loss of generality that $\mathrm{k}\leqslant r$.) Note that n and m are easily definable from the relations $\mathrm{f_s}$ and $\mathrm{f_p}$ in first order logic. Moreover, the natural ordering $\leqslant$ of A is definable from Sc by the FO(DTC) formula $[DTC_{u,v}\,Sc(u,v)](x,y)\vee x=y$. Thus, we can freely use the symbols n, m and $\leqslant$ in FO(DTC) and FO(TC) formulas, even though they are not contained in the vocabulary of table models. In the sequel we shall often regard Sc as a function; thus $Sc(x)$ denotes the successor of x.

We shall next introduce a few abbreviations which are useful in writing the formulas that express approximate pattern matching problems. Please note that we denote by *plain symbols* $s,t,\ldots,z$ elements in A, while the overlined versions are used for tuples $\bar{s},\bar{t},\ldots,\bar{z}\in A^3$.

- $\triangle(\bar{p},\bar{q})\Leftrightarrow((q_1=p_1)\wedge(q_2=Sc(p_2)))$,
- $\triangleleft(\bar{p},\bar{q})\Leftrightarrow((q_2=p_2)\wedge(q_1=Sc(p_1)))$,
- $\diamond(\bar{p},\bar{q})\Leftrightarrow((q_1=Sc(p_1))\wedge(q_2=Sc(p_2)))$.

The intuitive meaning of $\triangle(\bar{p},\bar{q})$ is that $\bar{p}$ represents the cell immediately above the cell $\bar{q}$ in a dynamic programming table. Similarly $\triangleleft(\bar{p},\bar{q})$ ($\diamond(\bar{p},\bar{q})$) says that $\bar{p}$ is the previous cell on the same row (on the same diagonal, respectively) as the cell $\bar{q}$.

3.3. Expressibility of approximate pattern matching

In our model theoretic framework, the approximate pattern matching problems can now be identified with the sets of models that correspond to them: Let $\mathscr{T}_{\mathrm{m}}$ be the set of all table models $\mathscr{A}_{P,S,k}$ such that P has a k-mismatches occurrence within S. Similarly, let $\mathscr{T}_{\mathrm{d}}$ be the set of all table models $\mathscr{A}_{P,S,k}$ such that P has a k-differences occurrence within S. We say that the k-mismatches (k-differences) problem is expressible in a logic $\mathscr{L}$ if there is a sentence φ of $\mathscr{L}$ over the vocabulary of table models such that $\mathscr{A}_{P,S,k}\in\mathscr{T}_{\mathrm{m}}$ ($\mathscr{A}_{P,S,k}\in\mathscr{T}_{\mathrm{d}}$, respectively) if and only if $\mathscr{A}_{P,S,k}\models\varphi$ holds for all table models $\mathscr{A}_{P,S,k}$.

Note that the sets $\mathscr{T}_{\mathrm{m}}$ and $\mathscr{T}_{\mathrm{d}}$ cannot be definable in any logic, since they are not closed under isomorphisms. Hence, in some model theoretic considerations it is useful to replace these sets by their closures under isomorphisms: We denote by $\mathscr{K}_{\mathrm{m}}$ the class of all models $\mathscr{A}$ such that $\mathscr{A}\cong\mathscr{A}_{P,S,k}$ for some $\mathscr{A}_{P,S,k}\in\mathscr{T}_{\mathrm{m}}$. Similarly, we let $\mathscr{K}_{\mathrm{d}}$ be the class of all models $\mathscr{A}$ such that $\mathscr{A}\cong\mathscr{A}_{P,S,k}$ for some $\mathscr{A}_{P,S,k}\in\mathscr{T}_{\mathrm{d}}$.

Using the model classes $\mathscr{K}_{\mathrm{m}}$ and $\mathscr{K}_{\mathrm{d}}$ we can rephrase the definition of expressibility as follows: the k-mismatches (k-differences) problem is expressible in $\mathscr{L}$ if and only if the class $\mathscr{K}_{\mathrm{m}}$ ($\mathscr{K}_{\mathrm{d}}$, respectively) is definable in $\mathscr{L}$.

Expressing the approximate pattern matching problems in as simple and natural logics as possible is desirable, since this can be seen as a first step towards constructing query languages for string databases. In the next two sections we shall prove that both the k-mismatches and the k-differences problems are expressible in transitive closure logics. This is quite a satisfactory result, as, e.g., the SQL:1999 standard comprises an explicit transitive closure operator (see [5]). Hence, a query language for string databases that is able to express approximate pattern matching problems could be directly based on this new version of SQL.

4. Expressing k-mismatches

Let us first consider the more straightforward problem, that is, expressing a k-mismatches occurrence of a pattern. This case shows itself to be a special case of our formalism; it can be expressed by an FO(DTC) formula. In Fig. 1, we give an example of k-mismatches calculation with dynamic programming, when $P=\mathtt{abc}$, $S=\mathtt{ddadc}$ and $k=1$. The accepting path (ending at position 5) is emphasized.

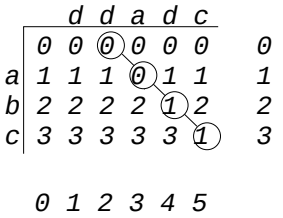

Fig. 1. Example of k-mismatches dynamic programming ($k=1$).

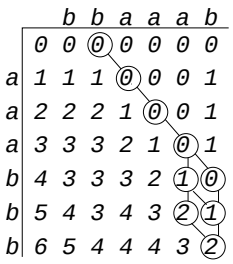

Fig. 2. Example of k-differences dynamic programming ($k=2$).

In this case a combination of Algorithms 1 and 2 is to be used:

Algorithm 3 (nondeterministic k-mismatches)
begin
1. $i := \mathtt{guess}(0, n)$;
2. $j := 0$; $count := 0$;
3. **repeat**
4. $j := j + 1$;
5. $count := count +$ (**if** $p_j = s_{i+j}$ **then** 0 **else** 1);
6. **until** $(j = m)$ **or** $(count > k)$;
7. **if** $count \leqslant k$ **then** *write* $(count, i + m)$;

end

Although Algorithm 3 is nondeterministic this nondeterminism is weak; the guess on line 1 can be expressed by using an existential quantifier.

Theorem 5. *The k-mismatches problem is expressible in* FO(DTC).

Proof. Let us start by giving an FO(DTC) sentence Φ_{m} that is established according to Algorithm 3. After that we analyze it. The sentence Φ_{m} is the following:

$$\exists \bar{x} \exists \bar{y}([DTC_{\bar{s}\bar{t}} \varphi_{\mathrm{m}}(\bar{s}, \bar{t})](\bar{x}, \bar{y}) \wedge (x_2 = 0) \wedge (x_3 = 0) \wedge (y_2 = m) \wedge (y_3 \leqslant k)), \quad (1)$$

where

$$\varphi_{\mathrm{m}}(\bar{s}, \bar{t}) = \exists u_1 \exists u_2 \exists v_1 \exists v_2 \quad (2)$$

$$((\diamond(\bar{s}, \bar{t})) \wedge \mathrm{f_p}(u_1, u_2) \wedge (u_1 = t_2) \wedge \mathrm{f_s}(v_1, v_2) \wedge (v_1 = t_1) \quad (3)$$

$$\wedge (((u_2 = v_2) \wedge (t_3 = s_3)) \vee (\neg(u_2 = v_2) \wedge (t_3 = Sc(s_3))))). \quad (4)$$

Line (1) represents the main part of the formula: it checks that $\bar{x}$ and $\bar{y}$ represent the two ends of an accepting path corresponding to a k-mismatches occurrence of the pattern. The transition formula $\varphi_{\mathrm{m}}(\bar{s}, \bar{t})$ is composed of the alignment (3) and action (4) parts. In the alignment part, at first, the advance direction is bound. In this case only the southeast direction is accepted $(\diamond(\bar{s}, \bar{t}))$, thus the path is always advanced in the considered diagonal. Then the associated characters within P and S are bound; bounding the character in P the horizontal alignment is used $(u_1 = t_2)$, analogously $(v_1 = t_1)$ aligns vertically the corresponding character in S. In the action part, the formula compares whether or not the corresponding characters u_2 of P and v_2 of S are the same and assigns a value for t_3 depending on the result of the comparison.

Note that the formula $\varphi_{\mathrm{m}}(\bar{s}, \bar{t})$ determines $\bar{t}$ uniquely from $\bar{s}$: $\varphi_{\mathrm{m}}(\bar{s}, \bar{t}) \wedge \varphi_{\mathrm{m}}(\bar{s}, \bar{t}') \rightarrow \bar{t} = \bar{t}'$. Thus, any φ_{m}-path is automatically non-branching.

Assume now that $\mathscr{A}_{P,S,k} \models \Phi_{\mathrm{m}}$. Then there are tuples $\bar{a}_0, \ldots, \bar{a}_m \in A^3$ such that $\bar{a}_0$ is of the form $(i, 0, 0)$, $\bar{a}_m$ is of the form (i', m, r) with $r \leqslant k$, and $\mathscr{A}_{P,S,k} \models \varphi_{\mathrm{m}}[\bar{a}_l, \bar{a}_{l+1}]$ holds for each $l < m$. From the above analysis we see that if $\bar{a}_l$ is a correct entry

(i,j,d_{ij}) in the dynamic programming table corresponding to $\mathscr{A}_{P,S,k}$, then $\bar{a}_{l+1}$ is the next correct entry $(i+1,j+1,d_{i+1,j+1})$ on the same diagonal. Since $\bar{a}_0$ is a correct entry on the top row, we conclude that the third component of $\bar{a}_m$ is $r=d_{i'm}$, which is k at the most. Therefore, $\mathscr{A}_{P,S,k}\in\mathscr{T}_{\mathrm{m}}$.

For the converse, assume that there is a minimizing path $d_{i,0},d_{i+1,1},\ldots,d_{i+m,m}$ in the dynamic programming table for P and S such that $d_{i+m,m}\leqslant k$. For each $l\leqslant m$, let $\bar{a}_l=(i+l,l,d_{i+l,l})$. Then it is clear that $\mathscr{A}_{P,S,k}\models\varphi_{\mathrm{m}}[\bar{a}_l,\bar{a}_{l+1}]$ for each $l<m$. As observed above, the φ_{m}-path $\bar{a}_0,\ldots,\bar{a}_m$ is necessarily non-branching, whence $\mathscr{A}_{P,S,k}\models[DTC_{\bar{s}\bar{t}}\varphi_{\mathrm{m}}(\bar{s},\bar{t})][\bar{a}_0,\bar{a}_m]$. As the other conjuncts of (1) are clearly satisfied by $\bar{a}_0$ and $\bar{a}_m$, we conclude that $\mathscr{A}_{P,S,k}\models\Phi_{\mathrm{m}}$. This completes our proof. □

5. Expressing k-differences

In the k-differences case the local transformations ID, E1–E3 are allowed. Allowing operations E2 and E3, as well, implies that in the dynamic programming table there may be several accepting paths starting from a cell d_{i0}. In Fig. 2 we give an example of a k-differences dynamic programming calculation, when $P=$ `aaabbb`, $S=$ `bbaaab`, and $k=2$. The accepting paths are emphasized. Since a minimizing path does not have to be unique anymore the nondeterministic `guess` function (recall it from Algorithm 2) is used to choose the correct advancing direction. An accepting path of k-differences calculation is next expressed by using a formula of FO(TC).

Theorem 6. *The k-differences problem is expressible in* FO(TC).

Proof. The proof is almost identical to the proof of the previous Theorem 5. We start by giving a sentence Φ_{d} of FO(TC), and then show that it captures the idea of the nondeterministic Algorithm 2 for k-differences. Let

$$\Phi_{\mathrm{d}}=\exists\bar{x}\exists\bar{y}([TC_{\bar{s}\bar{t}}\varphi_{\mathrm{d}}(\bar{s},\bar{t})](\bar{x},\bar{y})\wedge(x_2=0)\wedge(x_3=0)\wedge(y_2=m)\wedge(y_3\leqslant k)).$$

We repartition the action part of the formula into *condition* and *expense* parts, using a subscripted ψ and δ denoting the condition and expense parts, respectively. Now the transition formula is rewritten as follows:

$$\varphi_{\mathrm{d}}(\bar{s},\bar{t})=\exists u_1\exists u_2\exists v_1\exists v_2((\mathrm{f_p}(u_1,u_2)\wedge(u_1=t_2)\wedge\mathrm{f_s}(v_1,v_2)\wedge(v_1=t_1)) \tag{5}$$

$$\wedge((\psi_0\wedge\delta_0)\vee\ (\psi_1\wedge\delta_1)\vee\ (\psi_2\wedge\delta_2)\vee\ (\psi_3\wedge\delta_3))). \tag{6}$$

Line (5) fixes the vertical and horizontal alignments as before, while line (6) is associated with the actions. The condition parts are as follows.

$$\psi_0=((\diamond(\bar{s},\bar{t}))\wedge(u_2=v_2)),\quad \psi_2=(\triangleleft(\bar{s},\bar{t})),$$
$$\psi_1=((\diamond(\bar{s},\bar{t}))\wedge\neg(u_2=v_2)),\quad \psi_3=(\triangle(\bar{s},\bar{t})).$$

The formulas $\psi_0,\ldots,\psi_3$ correspond to the four possible steps on the path (cf. the local transformations ID, E1, E2 and E3). Furthermore, the use of an editing operation

should be charged (while the use of an identity should be free of charge). The expense formulas $\delta_0,\ldots,\delta_3$ corresponding to the (unit cost) local transformations ID, E1–E3, respectively, become as follows:

$$\delta_0 = (t_3 = s_3), \quad \delta_i = (t_3 = Sc(s_3)) \quad \text{for } i \in \{1,2,3\}.$$

It is now clear that $\mathscr{A}_{P,S,k} \in \mathscr{T}_{\mathrm{d}}$ if and only if Algorithm 2 accepts the input (P,S,k) if and only if $\mathscr{A}_{P,S,k} \models \varphi_{\mathrm{d}}$. □

Permitting individual costs for the transformations. In some applications it would be beneficial to have distinct costs for the local transformations. Denoting by $c(\mathrm{E}i)$ the cost associated with the local transformation Ei, any integer valued cost can be derived from the following inductive definition.

$$c(\mathrm{E}i) = 0 \iff \delta'_i = (t_3 = s_3) \tag{7}$$

$$c(\mathrm{E}i) = 1 \iff \delta'_i = (t_3 = Sc(s_3)) \tag{8}$$

$$\vdots$$

$$c(\mathrm{E}i) = n \iff \delta'_i = (t_3 = \underbrace{Sc(\ldots Sc(}_{n} s_3 \overbrace{)\ldots)}^{n+1}). \tag{9}$$

In order to use these costs, the expense parts (δ_i:s) of $\varphi_{\mathrm{d}}(\bar{s},\bar{t})$ (6) are substituted by the appropriate δ'_i:s.

We do not know whether the k-differences problem could be expressed in deterministic transitive closure logic already, but it seems to us that using FO(TC) instead of FO(DTC) is really necessary. However, it should be noted that proving that the k-differences problem is not expressible in FO(DTC) is at least as hard as separating NLOGSPACE from LOGSPACE.[4] This can be seen as follows. Recall that LOGSPACE computable properties of ordered models are expressible in FO(DTC). Since table models have an FO(DTC)-definable ordering, non-expressibility in FO(DTC) would imply that the k-differences problem is not in LOGSPACE. On the other hand, Theorem 6 implies that it is in NLOGSPACE.

6. Expressing k-differences when k is fixed

Let us now study the k-differences problem when k is fixed beforehand, i.e., k is not considered as a part of the input anymore. Henceforth we shall use the acronym *kdf* for the problem. The problem specification reflects slightly on the table models we use; the model $\mathscr{A}_{P,S}$ is otherwise identical to $\mathscr{A}_{P,S,k}$, but k is no more given as a constant. Note however that, since k is now a fixed integer, it can be defined by an FO formula (analogously to the inductive definition used on (9)).

[4] Whether the two classes are separate is an open problem in complexity theory.

In this case, the corresponding set of models is simply $\mathscr{T}_{\mathrm{d}}^{k} = \{\mathscr{A}_{P,S} \mid \mathscr{A}_{P,S,k} \in \mathscr{T}_{\mathrm{d}}\}$, and we say that the *kdf* problem is expressible in $\mathscr{L}$ if there is a sentence φ of $\mathscr{L}$ such that $\mathscr{T}_{\mathrm{d}}^{k} = \{\mathscr{A}_{P,S} \mid \mathscr{A}_{P,S} \models \varphi\}$. As in the previous cases, let $\mathscr{K}_{\mathrm{d}}^{k}$ be the closure of $\mathscr{T}_{\mathrm{d}}^{k}$ under isomorphisms.

From Theorem 6 it follows immediately that the *kdf* problem is expressible in FO(TC): $\mathscr{A}_{P,S} \in \mathscr{T}_{\mathrm{d}}^{k}$ if and only if $\mathscr{A}_{P,S} \models \Phi_{\mathrm{d}}^{k}$, where Φ_{d}^{k} is the sentence obtained from Φ_{d} by replacing the constant symbol for k by its definition in FO. Here we shall prove that the *kdf* problem is expressible already in FO(DTC).

In the proof we harness the fact that the accepting path must be in a diagonal band [20]. The width of such a band is restricted by k; the smaller the k, the narrower the band. The expressibility of this case in FO(DTC) is a consequence of the fact that the width of the band is known beforehand, and it suffices to compute the values of d_{ij} only relative to the diagonal band. More precisely, let d_{ij}^{B}, $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant m$, be defined as d_{ij} (according to Algorithm 1), except that d_{ij} is set to be n whenever the cell (i,j) is outside the diagonal band B in question. Then there is an accepting path in the original dynamic programming table if and only if for some diagonal band B there is $i \leqslant n$ such that $d_{im}^{B} \leqslant k$. Indeed, $d_{ij} \leqslant d_{ij}^{B}$ for all $i \leqslant n$ and $j \leqslant m$, whence $d_{im}^{B} \leqslant k$ guarantees the existence of an accepting path. On the other hand, if there is an accepting path, then $d_{ij} = d_{ij}^{B}$ holds for all cells (i,j) on the accepting path, where B is any diagonal band that contains the accepting path.

We shall cover the diagonal band with successive L-shaped polygons (see Fig. 3), which we call L-polies. We assume that $k < m$ (recall that $m < n$). Since we use unit cost local transformations, L-polies are always symmetric and they are comprised of $2k+1$ elements. The corner of an L-poly (the element $\#k+1$) is called the *pivot* of the L-poly (see Fig. 4). The elements within the ranges $1-k$ and $(k+2)-(2k+1)$ form the *right* and *left tails* of the L-poly, respectively.

As can also be seen in Fig. 3, either one of the two tails of an L-poly may exceed the border of the dynamic programming table. In order to avoid such overflows we use the following two additional successor functions (clearly they are definable in FO from the relations $\mathrm{f_s}$ and $\mathrm{f_p}$):

$$Sc_m(x) = \begin{cases} Sc(x) & \text{if } x < m, \\ x & \text{else.} \end{cases} \qquad Sc_n(x) = \begin{cases} Sc(x) & \text{if } x < n, \\ x & \text{else.} \end{cases} \tag{10}$$

By using the functions above we might get a degenerate form of such a polygon, which we still call L-poly.

Moreover, in order to avoid the problems with complicated indices, we change the notation slightly: the 'coordinates' of a dynamic programming table cell are denoted by a pair α, β (corresponding to x_1 and x_2 of the previous notation), respectively, while the value $d_{\alpha\beta}$ in that particular location is denoted by γ (corresponding to the previous x_3). Furthermore, let vectors with a 'hat' (e.g. $\hat{x}$) denote tuples of the form $((\alpha_1^{\hat{x}}, \beta_1^{\hat{x}}, \gamma_1^{\hat{x}}), \ldots, (\alpha_{2k+1}^{\hat{x}}, \beta_{2k+1}^{\hat{x}}, \gamma_{2k+1}^{\hat{x}}))$.

Lemma 7. *Let k be fixed. There is an* FO *formula $\varphi_L(\hat{x})$ such that $\mathscr{A}_{P,S} \models \varphi_L[\hat{a}]$ if and only if $\hat{a}$ is an L-poly in $\mathscr{A}_{P,S}$.*

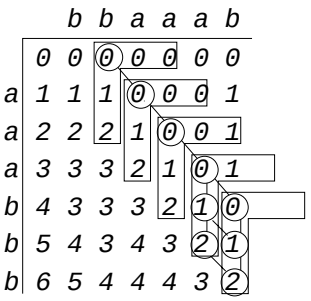

Fig. 3. Covering the band with L-polies.

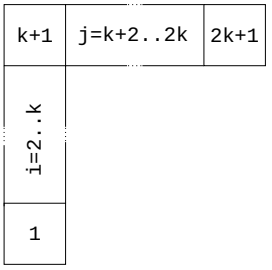

Fig. 4. The numeration of elements in an L-poly.

Proof. Let $\varphi_L(\hat{x})$ be the formula

$$\varphi_L(\hat{x}) = \left(\bigwedge_{i=1}^{k} (\alpha_i^{\hat{x}} = \alpha_{i+1}^{\hat{x}}) \wedge (\alpha_{k+i+1}^{\hat{x}} = Sc_n(\alpha_{k+i}^{\hat{x}})) \right.$$
$$\left. \wedge (\beta_i^{\hat{x}} = Sc_m(\beta_{i+1}^{\hat{x}})) \wedge (\beta_{k+i+1}^{\hat{x}} = \beta_{k+i}^{\hat{x}}) \right).$$

Clearly it satisfies the claim. □

Let us next define the successor relation for L-polies in our model. Let $\hat{a}$ be an L-poly whose pivot is not on the last row or last column of the dynamic programming table. The pivot of its successor $\hat{b}$ lies then on the same diagonal as the pivot of $\hat{a}$, at the adjacent southeast cell. Moreover, the γ values $\gamma_i^{\hat{b}}$ of the successor are dictated by the γ values of $\hat{a}$ according to the modified dynamic programming rules for computing $d_{\alpha\beta}^{B}$, where B is the diagonal band that covers the L-polies $\hat{a}$ and $\hat{b}$.

Definition 8. Let $\hat{a}$ and $\hat{b}$ be L-polies. We use here the following notation:

$$\lambda_i = \begin{cases} n & \text{for } i = 1, \\ \gamma_{i-1}^{\hat{a}} & \text{for } 2 \leqslant i \leqslant k+1, \\ \gamma_{i-1}^{\hat{b}}, & \text{for } k+2 \leqslant i \leqslant 2k+1. \end{cases} \qquad \mu_i = \begin{cases} \gamma_{i+1}^{\hat{b}} & \text{for } 1 \leqslant i \leqslant k, \\ \gamma_{i+1}^{\hat{a}} & \text{for } k+1 \leqslant i \leqslant 2k, \\ n & \text{for } i = 2k+1. \end{cases}$$

Then $\hat{b}$ is the successor of $\hat{a}$ in $\mathscr{A}_{P,S}$ if and only if the following conditions hold:

- $\alpha^{\hat{a}}_{k+1} < n$ and $\beta^{\hat{a}}_{k+1} < m$;
- $\alpha^{\hat{b}}_i = Sc_n(\alpha^{\hat{a}}_i)$ for each i, $1 \leqslant i \leqslant 2k+1$;
- $\beta^{\hat{b}}_i = Sc_m(\beta^{\hat{a}}_i)$ for each i, $1 \leqslant i \leqslant 2k+1$;
- $\gamma^{\hat{b}}_i = \min\{\gamma^{\hat{a}}_i + (\textbf{if } p_j = s_l \textbf{ then } 0 \textbf{ else } 1), \lambda_i + 1, \mu_i + 1\}$ for each i, $1 \leqslant i \leqslant 2k+1$, where $l = \alpha^{\hat{b}}_i$ and $j = \beta^{\hat{b}}_i$.

Clearly the conditions in the definition above guarantee that each L-poly has a unique successor, unless its pivot is on the last row or last column of the dynamic programming table (in which case the L-poly has no successors). It is also easy to see that this successor relation is definable in first order logic:

Lemma 9 (L-polies' successor relation). *Let k be fixed and let $\hat{a}$ and $\hat{b}$ be L-polies. There is an* FO *formula $\varphi_{Sc}(\hat{x},\hat{y})$ such that $\mathscr{A}_{P,S} \models \varphi_{Sc}[\hat{a},\hat{b}]$ if and only if $\hat{b}$ is the successor of $\hat{a}$.*

Proof. Let $\psi(\hat{x},\hat{y})$ be the formula

$$\psi(\hat{x},\hat{y}) = \alpha^{\hat{x}}_i < n \wedge \beta^{\hat{a}}_i < m \wedge \left(\bigwedge_{i=1}^{2k+1} (\alpha^{\hat{y}}_i = Sc_n(\alpha^{\hat{x}}_{k+1})) \wedge (\beta^{\hat{y}}_{k+1} = Sc_m(\beta^{\hat{x}}_{k+1})) \right).$$

Then $\mathscr{A}_{P,S} \models \psi[\hat{a},\hat{b}]$ if and only if the L-polies $\hat{a}$ and $\hat{b}$ satisfy the three first conditions of Definition 8. Similarly for each i, $1 \leqslant i \leqslant 2k+1$, there is an FO formula $\kappa_i(\hat{x},\hat{y})$ such that $\mathscr{A}_{P,S} \models \kappa_i[\hat{a},\hat{b}]$ if and only if $\hat{a}$ and $\hat{b}$ satisfy the corresponding instance of the last condition in Definition 8. Indeed, $\kappa_i(\hat{x},\hat{y})$ is the formula

$$\begin{aligned}\kappa_i(\hat{x},\hat{y}) = \exists u \exists v (&\mathrm{f_p}(\beta^{\hat{y}}_{k+1}, u) \wedge \mathrm{f_s}(\alpha^{\hat{y}}_{k+1}, v) \\ &\wedge ((u = v \wedge \theta_{\min}(\gamma^{\hat{x}}_i, Sc(\lambda_i), Sc(\mu_i), \gamma^{\hat{y}}_i)) \\ &\vee (\neg(u = v) \wedge \theta_{\min}(Sc(\gamma^{\hat{x}}_i), Sc(\lambda_i), Sc(\mu_i), \gamma^{\hat{y}}_i)))),\end{aligned}$$

where $\theta_{\min}(x,y,z,w)$ is a formula saying that $w = \min\{x,y,z\}$, and λ_i and μ_i are as in Definition 8 (with $\hat{x}$ and $\hat{y}$ in place of $\hat{a}$ and $\hat{b}$, respectively).

The desired formula $\varphi_{Sc}(\hat{x},\hat{y})$ is now obtained by taking the conjunction of the formulas ψ and κ_i, $1 \leqslant i \leqslant 2k+1$. □

Note that if $\hat{a}$ is an L-poly in a diagonal band B, $\hat{b}$ is its successor, and each $\gamma^{\hat{a}}_i$ is equal to the corresponding $d^B_{\alpha\beta}$, then the same holds for the γ-values of $\hat{b}$, too. Hence the modified dynamic programming table $d^B_{\alpha\beta}$ can be obtained by computing the deterministic path along the successor relation of L-polies. This is the key for proving our main result in this section:

Theorem 10. *The kdf problem is expressible in* FO(DTC).

Proof. The formula Φ_{f} defining the class $\mathscr{K}_{\mathrm{d}}^{k}$ is similar to the formulas Φ_{m} and Φ_{d} of the previous theorems:

$$\Phi_{\mathrm{f}} = \exists\hat{x}\exists\hat{y}\left([DTC_{\hat{s}\hat{t}}\varphi_{\mathrm{f}}(\hat{s},\hat{t})](\hat{x},\hat{y}) \wedge \varphi_L(\hat{x}) \wedge \varphi_L(\hat{y})\right. \tag{11}$$

$$\wedge \left(\bigwedge_{i=k+1}^{2k+1}(\beta_i^{\hat{x}} = \gamma_i^{\hat{x}} = 0)\right) \wedge \left(\bigwedge_{i=1}^{k}(\beta_i^{\hat{x}} = \gamma_i^{\hat{x}} = k+1-i)\right) \tag{12}$$

$$\left. \wedge \left(\bigvee_{i=1}^{2k+1}((\beta_i^{\hat{y}} = m) \wedge (\gamma_i^{\hat{y}} \leqslant k))\right)\right). \tag{13}$$

This formula states that there is a deterministic path from an L-poly $\hat{x}$ to another L-poly $\hat{y}$ along the relation defined by the formula φ_{f}. The conjuncts on the second line (12) ascertain that the pivot of $\hat{x}$ is on the topmost row of the dynamic programming table, and that each $\gamma_i^{\hat{x}} = d^B_{\alpha\beta}$, where $\alpha = \alpha_i^{\hat{x}}$, $\beta = \beta_i^{\hat{x}}$ and B is the diagonal band containing $\hat{x}$. Furthermore, the big disjunction on line (13) states that the stopping condition of an accepting path is met by one of the elements of the L-poly $\hat{y}$.

The formula $\varphi_{\mathrm{f}}(\hat{s},\hat{t})$ inside the *DTC* operator simply states that $\hat{s}$ and $\hat{t}$ are L-polies, and $\hat{t}$ is the successor of $\hat{s}$:

$$\varphi_{\mathrm{f}}(\hat{s},\hat{t}) = \varphi_L(\hat{s}) \wedge \varphi_L(\hat{t}) \wedge \varphi_{Sc}(\hat{s},\hat{t}).$$

As observed above, the γ-values on a deterministic path along the successor relation of L-polies are equal to the corresponding entries $d^B_{\alpha\beta}$ in the modified dynamic programming table, provided that this holds for the first L-poly in the topmost row. Hence we conclude that $\mathscr{A}_{P,S} \models \Phi_{\mathrm{f}}$ if and only if there is a diagonal band B and $\alpha \leqslant n$ such that $d^B_{\alpha m} \leqslant k$. As explained earlier, the latter condition is equivalent to $\mathscr{A}_{P,S} \in \mathscr{T}_{\mathrm{d}}^{k}$. □

7. On the complexity of pattern matching problems

In this section we derive both upper and lower bounds for the complexity of the approximate pattern matching problems. We shall first consider the problems in the usual complexity theoretic setting in which instances are strings that are directly used as inputs to Turing machines. Then we shall study the complexity of their model theoretic versions (i.e., the complexity of the model classes $\mathscr{K}_{\mathrm{m}}$, $\mathscr{K}_{\mathrm{d}}$ and $\mathscr{K}_{\mathrm{d}}^{k}$).

It is important to note that the complexity of the problems in these two different settings is not necessarily the same. Indeed, we shall show that $\mathscr{K}_{\mathrm{m}}$ is LOGSPACE complete, which is probably not true for the k-mismatches problem in the usual setting. The reason for this difference is that in the model theoretic setting the natural order of the strings P and S is no more available for free in the Turing machine computations; rather, the input model (or its adjacency matrix) is encoded as a binary string in an arbitrary order, and the natural order of the strings P and S has to be computed from this encoding.

We also consider another model theoretic formulation of the approximate pattern matching problems in which strings are represented by linear orders instead of successor relations. With this modification, computing the natural order of the strings P and S

from the encoding of the input model becomes significantly easier. This is reflected in the complexity of the problems: we shall show that in this modified model theoretic setting the k-mismatches problem is already in the class ThC^0.

7.1. Upper bounds for the algorithms

We start by comparing the efficiency of the approximate pattern matching algorithms that we have considered in the paper. Since Algorithm 2 is nondeterministic, a direct comparison with the deterministic Algorithm 1 is actually not possible. However, for space complexity we can make such a comparison indirectly by using Savitch's theorem, which relates nondeterministic space to deterministic space:

Fact 11 (Savitch [18]). *A nondeterministic $s(n)$-space bounded Turing machine can be simulated by a deterministic $(s(n))^2$-space-bounded Turing machine, provided $s(n) \geqslant \log n$.*

Recall that with respect to the RAM model of computation, the space requirement of Algorithm 2 is O(1). However, before applying Fact 11 we have to find the space complexity with respect to the Turing model of computation. This is straightforward: Storing the values of the variables i, j and *count* requires $O(\log n)$ space, $O(\log m)$ space and $O(\log k)$ space, respectively. Hence the total space requirement is $O(\log n)$ (recall that we are assuming that $k, m \leqslant n$).

By Fact 11, Algorithm 2, which includes both of the k-differences cases (k is part of the input, and fixed k), can be simulated deterministically in $O((\log n)^2)$ space. This improves the upper bound $O(m)$ on the required space of Algorithm 1 for inputs in which the length of the pattern is relatively large compared to the length of the text, e.g., when $m \geqslant n^{\varepsilon}$ for some $\varepsilon > 0$.

In the case of the k-mismatches problem, we can get even better space efficiency. To be specific, it is easy to modify Algorithm 3 so that it becomes deterministic: just replace the guess (on line 1) by a repeat-until loop. Clearly, the space requirement of this modified algorithm is the same as that of Algorithm 2, and hence we obtain the space bound $O(\log n)$.

Summing up the reasoning above we get

Proposition 12.
(1) *The k-mismatches problem is in* LOGSPACE.
(2) *The k-differences problem is in* NLOGSPACE, *and consequently in* $\mathrm{DSPACE}((\log n)^2)$.

7.2. Complexity in the model theoretic setting

We consider next the complexity of the model classes $\mathscr{K}_{\mathrm{m}}$, $\mathscr{K}_{\mathrm{d}}$ and $\mathscr{K}_{\mathrm{d}}^k$. The first observation is that we get the analogue of Proposition 12 as a direct corollary from Theorems 5 and 6:

Proposition 13.
(1) $\mathscr{K}_{\mathrm{m}}$ *is in* LOGSPACE
(2) $\mathscr{K}_{\mathrm{d}}$ *is in* NLOGSPACE, *and consequently in* DSPACE($(\log n)^2$).

Moreover, from Theorem 10 we get

Proposition 14. (3) $\mathscr{K}_{\mathrm{d}}^{k}$ *is in* LOGSPACE.

For the class $\mathscr{K}_{\mathrm{m}}$ we can prove that the upper bound given in Propositions 13 is the best possible: there is a LOGSPACE complete problem that is reducible to it. Let $\mathscr{K}_{\mathrm{ord}}$ be the set of models $\langle A, Sc, a, b\rangle$ such that Sc is successor relation on A, and $a<b$ with respect to the corresponding linear order $\leqslant$. Note that the problem $\mathscr{K}_{\mathrm{ord}}$ is the restriction of deterministic reachability 'given $R \subseteq A^2$ and $a, b \in A$, is $(a,b) \in \mathrm{DTC}(R)$' to the case of successor relations. In [6], Etessami proved that $\mathscr{K}_{\mathrm{ord}}$ is LOGSPACE complete with respect to (quantifier free) first order reductions (see [9] for a definition of first order reductions).

Lemma 15. *The class* $\mathscr{K}_{\mathrm{ord}}$ *is reducible to* $\mathscr{K}_{\mathrm{m}}$ *via a first order reduction.*

Proof. The reduction of $\mathscr{K}_{\mathrm{ord}}$ to $\mathscr{K}_{\mathrm{m}}$ goes as follows. Let $\langle A, R, a, b\rangle$ be an instance of the problem $\mathscr{K}_{\mathrm{ord}}$; we shall associate with it an instance $\langle A, R^{a,b}, F_{\mathrm{p}}, F_{\mathrm{s}}, k\rangle$ of $\mathscr{K}_{\mathrm{m}}$. We may assume that there are unique R-minimal and R-maximal elements min and max, and that each element $x \neq \max$ or $x \neq \min$ has a unique R-successor or R-predecessor, respectively (note that this is clearly expressible by an FO formula); otherwise we let $R^{a,b} = R$, whence automatically $\langle A, R^{a,b}, F_{\mathrm{p}}, F_{\mathrm{s}}, k\rangle \notin \mathscr{K}_{\mathrm{m}}$. Let c and d be the R-successors of a and b, respectively. We define the relation $R^{a,b}$ from R by

$$R^{a,b} = (R \setminus \{(a,c),(b,d)\}) \bigcup \{(\max, c),(b, \min)\}.$$

Furthermore, we define F_{p} and F_{s} to be the constant function with value min: $F_{\mathrm{p}} = F_{\mathrm{s}} = \{(x, \min) \mid x \in A\}$. Finally, we let $k = \min$. It is easy to see that mapping $\langle A, R, a, b\rangle$ to $\langle A, R^{a,b}, F_{\mathrm{p}}, F_{\mathrm{s}}, \min\rangle$ is a first order reduction (i.e., the relations $R^{a,b}$, F_{p}, F_{s} and the constant min are definable from R, a and b by FO formulas).

We claim now that $\langle A, R, a, b\rangle \in \mathscr{K}_{\mathrm{ord}}$ if and only if $\langle A, R^{a,b}, F_{\mathrm{p}}, F_{\mathrm{s}}, \min\rangle \in \mathscr{K}_{\mathrm{m}}$. Indeed, if R is a successor relation and $a<b$ with respect to the corresponding order, then $R^{a,b}$ is also a successor relation. Since the pattern P and text S corresponding to F_{p} and F_{s} are identical, P has a k-mismatches occurrence within S irrespective of k. On the other hand, if R is not a successor relation, or if $b \leqslant a$ in the corresponding ordering, then the relation $R^{a,b}$ contains a cycle. Hence, $\langle A, R^{a,b}, F_{\mathrm{p}}, F_{\mathrm{s}}, \min\rangle$ cannot be isomorphic to any table model. □

Note that allowing the inserting and deleting operations E2 and E3 does not change anything in the argument above. Thus $\mathscr{K}_{\mathrm{ord}}$ is reducible to the k-differences problem as well; moreover, this holds even for the *kdf* problem. Thus, we get the following corollary.

Corollary 16.
(1) $\mathscr{K}_{\mathrm{m}}$ *is* LOGSPACE *complete.*
(2) $\mathscr{K}_{\mathrm{d}}$ *is* LOGSPACE *hard.*
(3) $\mathscr{K}_{\mathrm{d}}^{k}$ *is* LOGSPACE *complete.*

Corollary 16 settles the complexity of $\mathscr{K}_{\mathrm{m}}$ and $\mathscr{K}_{\mathrm{d}}^{k}$ completely, but leaves a gap in the case of $\mathscr{K}_{\mathrm{d}}$. It seems plausible to us that $\mathscr{K}_{\mathrm{d}}$ is neither in LOGSPACE, nor NLOGSPACE complete. But of course, this cannot be proved without a major breakthrough in complexity theory.

7.3. Complexity in alternative model theoretic setting

In the model theoretic framework we adopted in Section 3.2, strings are represented in table models by a successor relation. Another natural way of representing strings is to use linear order as a basic relation instead of successor relation. In this modified setting, table models are defined in the same way as in Section 3.2, except that the successor relation Sc is replaced by the natural ordering $\leqslant$ of the universe. We denote by $\mathscr{A}_{P,S,k}^{\leqslant}$ the model that is obtained from $\mathscr{A}_{P,S,k}$ in this way.

We shall concentrate here on the complexity of the k-mismatches problem in this alternative model theoretic setting. We denote the corresponding set $\{\mathscr{A}_{P,S,k}^{\leqslant} \mid \mathscr{A}_{P,S,k} \in \mathscr{T}_{\mathrm{m}}\}$ of modified table models by $\mathscr{T}_{\mathrm{m}}^{\leqslant}$ and its closure under isomorphisms by $\mathscr{K}_{\mathrm{m}}^{\leqslant}$.

It turns out that there is an intimate connection between the class $\mathscr{K}_{\mathrm{m}}^{\leqslant}$ and the counting extension FO(COUNT) of first-order logic. The first observation is that $\mathscr{K}_{\mathrm{m}}^{\leqslant}$ is definable in FO(COUNT):

Theorem 17. *There is a sentence* φ *of* FO(COUNT) *such that* $\mathscr{T}_{\mathrm{m}}^{\leqslant} = \{\mathscr{A}_{P,S,k}^{\leqslant} \mid \mathscr{A}_{P,S,k}^{\leqslant} \models \varphi\}$.

Proof. We shall first write a formula $\theta(x,y)$ such that $\mathscr{A}_{P,S,k}^{\leqslant} \models \theta[a,b]$ if and only if $1 \leqslant a \leqslant m$, $1 \leqslant b+a \leqslant n$ and $p_a \neq s_{b+a}$, where $P = p_1 \ldots p_m$ and $S = s_1 \ldots s_n$. Indeed, let $\chi(x,y,z)$ be the formula

$$\exists i\,(\exists^{=i}u\,(u < x) \wedge \exists^{=i}u\,(y \leqslant u < z)).$$

Then $\chi(x,y,z)$ says that $y + x = z$, whence

$$\theta(x,y) := \exists z\,(\chi(x,y,z) \wedge \exists u \exists v(\mathrm{f_p}(x,u) \wedge \mathrm{f_s}(z,v) \wedge u \neq v))$$

is as desired.

Now the formula

$$\psi(y,i) := \exists z\,(\chi(m,y,z) \wedge z \leqslant n) \wedge \exists^{=i}x\,\theta(x,y)$$

says that $y+m \leqslant n$ (so that the factor $s_{y+1} \ldots s_{y+m}$ of S exists) and $D_H(P, s_{y+1} \ldots s_{y+m}) = i$. Hence we conclude that $\mathscr{A}_{P,S,k}^{\leqslant} \in \mathscr{T}_{\mathrm{m}}^{\leqslant}$ if and only if $\mathscr{A}_{P,S,k}^{\leqslant} \models \varphi$, where φ is the sentence $\exists y \exists i\,(\exists^{i}x\,(x < k) \wedge \psi(y,i))$. $\square$

Corollary 18. *$\mathscr{K}_{\mathrm{m}}^{\leqslant}$ is in* ThC^0.

This result shows that there is a sharp difference in the complexity of $\mathscr{K}_{\mathrm{m}}^{\leqslant}$ and $\mathscr{K}_{\mathrm{m}}$: as $\mathscr{K}_{\mathrm{m}}$ is LOGSPACE complete, it is not in ThC^0, unless LOGSPACE collapses to ThC^0.

On models that are equipped with order and arithmetics we can prove a converse of Theorem 17. Let $\mathrm{FO}(Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}})$ be the extension of first order logic by the generalized quantifier $Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}}$ corresponding to the class $\mathscr{K}_{\mathrm{m}}^{\leqslant}$. That is, $\mathrm{FO}(Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}})$ is the extension of FO with the new formula formation rule:

- $Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}} x_1x_2, y_1y_2, z_1z_2, u\,(\varphi(x_1,x_2), \psi_{\mathrm{p}}(y_1,y_2), \psi_{\mathrm{s}}(z_1,z_2), \theta(u))$ is a formula, if $\varphi, \psi_{\mathrm{p}}, \psi_{\mathrm{s}}$ and θ are formulas.

The semantics of the quantifier $Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}}$ is given by the rule

- $\mathscr{A} \models Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}} x_1x_2, y_1y_2, z_1z_2, u\,(\varphi(x_1,x_2), \psi_{\mathrm{p}}(y_1,y_2), \psi_{\mathrm{s}}(z_1,z_2), \theta(u))$ if and only if $\langle A, \varphi^{\mathscr{A}}, \psi_{\mathrm{p}}^{\mathscr{A}}, \psi_{\mathrm{s}}^{\mathscr{A}}, a\rangle \in \mathscr{K}_{\mathrm{m}}^{\leqslant}$ for all $a \in \theta^{\mathscr{A}}$, where $\varphi^{\mathscr{A}} = \{(a_1,a_2) \in A^2 \mid \mathscr{A} \models \varphi[a_1,a_2]\}$ etc.

We refer the reader to [4] for more details on logics with generalized quantifiers. Let us just mention here that $\mathrm{FO}(Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}})$ is the least extension of FO in which $Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}}$ is definable and which is closed under first order operations and substituting relations by formulas. In particular, since FO(COUNT) has these closure properties, Theorem 17 implies that $\mathrm{FO}(Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}}) \leqslant \mathrm{FO(COUNT)}$. That is, for every sentence φ of $\mathrm{FO}(Q^{\leqslant}_{\mathscr{K}_{\mathrm{m}}})$, there is a sentence φ' of FO(COUNT) such that $\mathscr{A} \models \varphi \Leftrightarrow \mathscr{A} \models \varphi'$ holds for every $\mathscr{A}$.

We call a model $\mathscr{A}$ arithmetical, if it has relations $R^{\mathscr{A}}_{\leqslant}$, $R^{\mathscr{A}}_{+}$ and $R^{\mathscr{A}}_{\times}$ such that $\langle A, R^{\mathscr{A}}_{\leqslant}, R^{\mathscr{A}}_{+}, R^{\mathscr{A}}_{\times}\rangle$ is isomorphic to an initial segment of natural numbers equipped with its natural ordering and (the restrictions of) the addition and multiplication relations.

Theorem 19. *On arithmetical models,* $\mathrm{FO(COUNT)} \leqslant \mathrm{FO}(Q^{\leqslant_{\mathrm{m}}}_{\mathscr{K}})$.

Proof. Note first that on arithmetical models FO(COUNT) can be formulated without using the additional second sort: the second sort would just be an isomorphic copy of the restriction of the first sort to the linear order, addition and multiplication relations. Thus, we shall assume here that FO(COUNT) formulas contain only first sort variables.

Let $\varphi(\bar{x})$ be a formula of FO(COUNT). We prove by induction on φ that there is a formula $\varphi'(\bar{x})$ of $\mathrm{FO}(Q_{\mathscr{K}_{\mathrm{m}}^{\leqslant}})$ which is equivalent to it on all arithmetical models. That is, for every arithmetical model $\mathscr{A}$ and every $\bar{a} \in A^n$, $\mathscr{A} \models \varphi[\bar{a}] \Leftrightarrow \mathscr{A} \models \varphi'[\bar{a}]$.

If φ is an atomic formula, we let $\varphi' = \varphi$; the equivalence of φ and φ' is then trivial. The induction steps for first order connectives and quantifiers are also trivial: we simply let $(\neg\varphi)' = \neg\varphi'$, $(\varphi \wedge \psi)' = \varphi' \wedge \psi'$, and $(\exists x\varphi)' = \exists x\varphi'$.

Consider then the induction step for the counting quantifier $\exists^y z$. Thus, assume that $\varphi = \varphi(\bar{x}, y)$ is of the form $\exists^y z\, \psi(\bar{x}, z)$. The idea here is that we write formulas $\theta_P(u_1,u_2)$ and $\theta_S(v_1,v_2)$ which define on each arithmetical model two strings P and S such that $D_H(P,S)$ is exactly the number of elements z satisfying $\psi(\bar{x},z)$. Then $\varphi(\bar{x},y)$ is true if and only if P does not have a k-mismatches occurrence in S, where k is the predecessor of y in the linear order. Hence, $\varphi(\bar{x},y)$ is equivalent to the formula $\neg Q_{\mathscr{K}_{\mathrm{m}}^{\leqslant}} x_1x_2, u_1u_2, v_1v_2, w\,(x_1 \leqslant x_2, \theta_P(u_1,u_2), \theta_S(v_1,v_2), w+1=y)$.

It remains to define suitable formulas $\theta_P(u_1,u_2)$ and $\theta_S(v_1,v_2)$. This is straightforward; e.g.,

$$\theta_P(u_1,u_2) := (u_2 = \min)$$

and

$$\theta_S(v_1,v_2) := (\neg\psi(\bar{x},v_1)' \wedge v_2 = \min) \vee (\psi(\bar{x},v_1)' \wedge v_2 = \max)$$

are as desired (min and max are the definable constants referring to the least and the greatest element in the linear order). □

By Theorems 17 and 19 the logic FO($Q_{\mathcal{K}_m^{\leqslant}}$) has exactly the same expressive power as FO(COUNT) on arithmetical models. Consequently, FO($Q_{\mathcal{K}^{\leqslant}}$) captures the complexity class ThC^0 on the class of arithmetical models. Note that this does not mean that $\mathcal{K}_m^{\leqslant}$ would be a complete problem for ThC^0, but nevertheless it shows that in a natural model theoretic framework, the lower bound result given in Corollary 18 cannot be improved.

8. A case study: music information retrieval

Let us now consider MIR as a case study for the formalism. First we briefly describe the features specific to this application, then we show that our formalism can be adapted to meet the requirements of the application. Originally, the general string matching methods have been developed to strings over alphabets where there is no distance nor a ratio defined between the elements of the alphabet. In MIR applications, usually a simplified representation of (monophonic) music that gives only the pitch levels of the notes is used. These pitch levels can be given as integer values (e.g. as in MIDI). Hence a piece of music containing n notes is represented as a sequence of n integers (see Fig. 5). We say that strings encoding music by its pitch values use *absolute representation*. Since the alphabet now comprises integers, distances between the elements become definable.

Definition 20. Let $\bar{A}$ and A be strings of $\mathbb{Z}^*$. $\bar{A}=\bar{a}_1\cdots\bar{a}_n$ is the *interval representation* of $A=a_1\cdots a_n$, if $\bar{a}_1=a_1$ and $\bar{a}_i=a_i-a_{i-1}$, for $i=2,\ldots,n$.

Note that the interval representation is *lossless* because $\bar{a}_1$ stores the first value of A ($\bar{a}_1$ is usually considered as a *don't care* value in the matching). Therefore, a found occurrence of the pattern may represent a lower or a higher pitch sequence than that of the pattern, thus *transposition invariance* is obtained (see e.g. [13]).

To detect a single incorrect pitch value in a sequence using interval representation, the following generalization of operation (E4) should be used [13].

Definition 21 (E5). Let $\bar{a}_{j-1}\bar{a}_j$ and $\bar{b}_{i-1}\bar{b}_i$ be factors of $\bar{A}$ and $\bar{B}$ ($2\leqslant j\leqslant m$, $2\leqslant i\leqslant n$, $\bar{a}_j\neq\bar{b}_i$), respectively. In *compensation* $\bar{a}_{j-1}\bar{a}_j$ corresponds to $\bar{b}_{i-1}\bar{b}_i$, such that $\bar{a}_{j-1}+\bar{a}_j=\bar{b}_{i-1}+\bar{b}_i$. The corresponding compensation operation is: $\bar{a}_{j-1}\bar{a}_j\mapsto\bar{b}_{i-1}\bar{b}_i$.

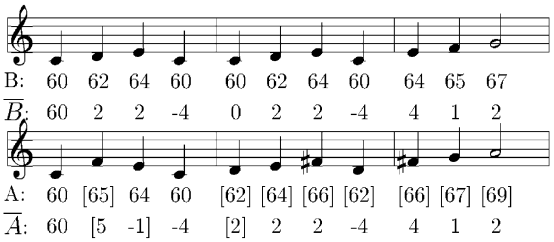

Fig. 5. Absolute vs. interval representation. The absolute representation corresponds to a sequence of MIDI pitches.

Remark 22. When using interval representation, the original transposition operation (E4) is a special case of the compensation operation (E5).

Let us denote by $\bar{D}(\bar{A},\bar{B})$ edit distance calculation using interval representation and permitting the use of operation E5. In Fig. 5, one can see the advantage of this distance measure in this application. We have enclosed in brackets elements requiring a local transformation. Note that A is rather similar to B; it contains one incorrect pitch and one *modulation*.[5] If the interval representation is used, the incorrect pitch can be detected by compensation ($\bar{a}_2+\bar{a}_3=\bar{b}_2+\bar{b}_3=4$) and the modulation by replacing. Thus the edit distance becomes $\bar{D}(\bar{A},\bar{B})=2$. However, if absolute representation is used the incorrect pitch is detected by a single replacing but the modulation has shifted all the remaining values ($D(A,B)=8$).

Let us now show that even this MIR application, using interval encoding and including the novel compensation operation, is expressible in our formalism. We need to be able to express the addition operation. Therefore, we add a ternary relation symbol $\oplus$ to the vocabulary of table models, and define its interpretation as $\{(\sigma(a),\sigma(b),\sigma(c))\in A^3 \mid a,b,c\in\Sigma, a+b=c\}$. Moreover, we assign a cost of 1 for (E5): $\delta_5=\ (t_3=Sc(s_3))$. Note that $(\psi_5\wedge\delta_5)$ should be included at the end of formula $\varphi_{\mathrm{d}}(\bar{s},\bar{t})$ (5), (6) with a disjunction operator. The condition part for the compensation operation is as follows:

$$\psi_5 = \exists i_1\exists i_2\exists j_1\exists j_2\exists\bar{h}((\diamond(\bar{s},\bar{h}))\wedge(\diamond(\bar{h},\bar{t})) \tag{14}$$

$$\wedge(u_1=Sc(i_1))\wedge(v_1=Sc(j_1))\wedge\exists w(\oplus(u_2,i_2,w)=\oplus(v_2,j_2,w))), \tag{15}$$

where $(\diamond(\bar{s},\bar{h}))\wedge(\diamond(\bar{h},\bar{t}))$ binds the predecessor of the considered element to be the second neighbour in the northwest direction. Pairs (i_1,i_2) and (u_1,u_2) correspond to the elements of the pattern, (j_1,j_2) and (v_1,v_2) to the elements of the text that are under consideration.

[5] A transposition that occurs anywhere else but at the beginning of a string.

9. Conclusion

We have considered the problem of expressing approximate pattern matching in strings by using transitive closure logics. Our formalism simulates the calculation along an accepting path of dynamic programming approach.

When k is given as a part of the input, the basic problem of k-mismatches can be expressed straightforwardly by FO(DTC). When proceeding to a more challenging problem of k-differences, we have also proceeded to more expressive logic, that is FO(TC). It is not certain, however, that this problem could not be expressed by FO(DTC), but proving that such a formula does not exist is at least as difficult as to show that NLOGSPACE$\neq$LOGSPACE.

Moreover, we have proven that even the k-differences problem can be expressed by FO(DTC), if the k is fixed beforehand. In that formalism we harnessed the fact that the minimizing path has to be in a diagonal band whose width is restricted by k, and cover the band with successive symmetrical polygons.

The obtained formalisms can be used both practically and theoretically. The practical advantage of the formalism is that a query language to string databases based on the SQL:1999 standard or the relation algebra **A**, can be developed. Thus, our formalism can be used as basis in building up a query engine for a string database (for instance, for content-based music retrieval). The theoretical advantage is that we were able to derive complexity bounds (with respect to the Turing model of computation) for the considered problems. The upper space complexity bound of the k-differences problem became $O((\log n)^2)$. For the case of the k-mismatches problem, this upper bound was improved to $O(\log n)$. On the other hand, we proved that in our model theoretic formalism, the latter cannot be improved, since there is a LOGSPACE complete problem that is reducible to the corresponding class of models. However, this lower bound is sensitive to the way the problem is formulated: we proved that after a small modification in the formalism, the k-mismatches problem becomes expressible in the logic FO(COUNT), whence the problem is already in ThC^0.

References

[1] A. Brazma, I. Jonassen, E. Ukkonen, J. Vilo, Discovering patterns and subfamilies in biosequences, in: Proc. 4th Internat. Conf. on Intelligent Systems for Molecular Biology, ISMB '96, AAAI Press, 1996, pp. 34–43.
[2] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, Oxford, 1994.
[3] C.J. Date, H. Darwen, The Third Manifesto, Addison-Wesley, New York, 1998.
[4] H.-D. Ebbinghaus, J. Flum, Finite Model Theory, Springer, Berlin, Heidelberg, 1997.
[5] A. Eisenberg, J. Melton, SQL:1999, formerly known as SQL3, SIGMOD Record 28 (1) (1999) 131–138.
[6] K. Etessami, Counting quantifiers, successor relations, and logarithmic space, J. Comput. System Sci. 54 (1997) 400–411.
[7] G. Grahne, M. Nykänen, E. Ukkonen, Reasoning about strings in databases, in: Proc. 13th ACM SIGACT-SIGMOD-SIGART symp. on Principles of database systems (PODS'94), 1994, pp. 303–312.
[8] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, Cambridge, 1997.
[9] N. Immerman, Descriptive Complexity, Springer, New York, 1999.

[10] G.M. Landau, U. Vishkin, Fast string matching with k differences, J. Comput. Systems 37 (1988) 63–78.
[11] K. Lemström, String matching techniques for music retrieval, Ph.D. Thesis, University of Helsinki, Department of Computer Science, 2000, Report A-2000-4.
[12] K. Lemström, L. Hella, Approximate pattern matching is expressible in transitive closure logic, in: Proc. 15th Ann. Symp. on Logic in Computer Science (LICS'2000), Santa Barbara, CA, June 2000, pp. 157–167.
[13] K. Lemström, E. Ukkonen, Including interval encoding into edit distance based music comparison and retrieval, in: Proc. AISB'2000 Symp. on Creative & Cultural Aspects and Applications of AI & Cognitive Science, Birmingham, April 2000, pp. 53–60.
[14] MIDI Manufacturers Association, Los Angeles, CA, The Complete Detailed MIDI 1.0 Specification, 1996.
[15] G. Myers, A fast bit-vector algorithm for approximate string matching based on dynamic programming, J. ACM 46 (1999) 395–415.
[16] M. Nykänen, Querying string databases with modal logic, Ph.D. Thesis, University of Helsinki, Department of Computer Science, 1997, Report A-1997-3.
[17] E.G.M. Petrakis, Image representation, indexing and retrieval based on spatial relationships and properties of objects, Ph.D. Thesis, University of Crete, Department of Computer Science, March 1993.
[18] W.J. Savitch, Relationship between nondeterministic and deterministic tape complexities, J. Comput. System Sci. 4 (1970) 177–192.
[19] P.H. Sellers, The theory and computation of evolutionary distances: pattern recognition, J. Algorithms 1 (4) (1980) 359–373.
[20] E. Ukkonen, Algorithms for approximate string matching, Inform. Control 64 (1985) 100–118.
[21] E. Ukkonen, Finding approximate patterns in strings, J. Algorithms 6 (1985) 132–137.
[22] P. van Emde Boas, Machine models and simulations, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, Elsevier, Amsterdam, 1990, pp. 3–66.
[23] P. Weiner, Linear pattern matching algorithms, in: Proc. IEEE 14th Annu. Symp. on Switching and Automata Theory, 1973, pp. 1–11.